In-memory Integration of Existing Software Components for Parallel Adaptive Unstructured Mesh Workflows

Cameron W. Smith
SCOREC
Rensselaer Polytechnic Institute
110 8th St., Troy NY 12180
smithc11@rpi.edu

Brian Granzow
SCOREC
Rensselaer Polytechnic Institute
110 8th St., Troy NY 12180
granzb@rpi.edu

Dan Ibanez
SCOREC
Rensselaer Polytechnic Institute
110 8th St., Troy NY 12180
ibaned@rpi.edu

Onkar Sahni
SCOREC
Rensselaer Polytechnic Institute
110 8th St., Troy NY 12180
sahni@rpi.edu

Kenneth E. Jansen
University of Colorado Boulder
1111 Engineering Dr.,
Boulder, CO 80309
jansenke@colorado.edu

Mark S. Shephard
SCOREC
Rensselaer Polytechnic Institute
110 8th St., Troy NY 12180
shephard@rpi.edu

ABSTRACT

Reliable mesh-based simulations are needed to solve complex engineering problems. Mesh adaptivity can increase reliability by reducing discretization errors, but requires multiple software components to exchange information. Often, components exchange information by reading and writing a common file format. This file-based approach becomes a problem on massively parallel computers where filesystem bandwidth is a critical performance bottleneck. Our data stream and component interface approaches avoid the filesystem bottleneck. In this paper we present our approaches and their use within the PHASTA computational fluid dynamics solver and Albany multiphysics framework. Information exchange performance results are reported on up to 2048 cores of a BlueGene/Q system.

Keywords

In-memory integration, Unstructured mesh adaptation, Parallel workflows

CCS Concepts

•**Computing methodologies** → **Massively parallel and high-performance simulations;** Simulation tools; •**Software and its engineering** → **Reusability;** •**Information systems** → Distributed storage; •**Applied computing** → Computer-aided design;

1. INTRODUCTION

Unstructured mesh methods, like finite elements and finite volumes, support the effective analysis of complex physical behaviors modeled by partial differential equations over general three-dimensional domains. The most reliable and efficient methods apply adaptive procedures with a-posteriori error estimators that indicate where and how the mesh is to be modified. Although adaptive meshes can have two to three orders of magnitude fewer elements than a more uniform mesh for the same level of accuracy, there are many complex simulations where the meshes required are so large that they can only be solved on massively parallel systems.

Simulations on massively parallel systems are most effective when data movement is minimized. Data movement costs increase with the depth of the memory hierarchy; a design trade-off for increased capacity. For example, the highest level on-node storage in the IBM BlueGene/Q A2 processor [16] is the per-core 16KiB L1 cache (excluding registers) and has a peak bandwidth of 819 GiB/s. The lowest level on-node storage, 16GiB of DDR3 main memory, provides a million times more capacity at the cost of 19 times less bandwidth, 43GiB/s [24]. One level further down the hierarchy is the parallel filesystem[1]. At this level, the bandwidth and capacity relationship is less clear as the filesystem is a shared resource. Table 1 lists the per-node peak main memory and filesystem bandwidth across four generations of Argonne National Laboratory leadership class systems: BlueGene/L [38, 3], Intrepid BlueGene/P [23, 4], Mira BlueGene/Q [16, 10], and 2018's Aurora [19]. Based on these peak values the bandwidth gap between main memory and the filesystem is at least three orders of magnitude. Software must leverage the cache and main memory bandwidth performance advantage during as many operations as possible to maximize performance.

The core interfaces supporting adaptive workflows are detailed in Section 2. Section 3 reviews methods to couple workflow software. Section 4 details the data stream approach to avoid filesystem use. Section 5 details the use

[1]We assume, for the sake of simplicity, that the main memory of other nodes is not available. In practice, most applications do not not use all the on-node memory and some checkpoint-restart methods take advantage of this for increased performance [9, 20, 21].

XSEDE16, July 17 - 21, 2016, ,

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-4755-6/16/07...$15.00

DOI: http://dx.doi.org/10.1145/2949550.2949650

Table 1: Per-node main memory and filesystem peak bandwidth over four generations of Argonne National Laboratory systems. The values in parentheses indicate the increase relative to the previous generation system.

	Memory BW (GiB/s)	Filesystem BW (GiB/s)
BG/L	5.6	0.0039
BG/P	14 (2.4x)	0.0014 (0.36x)
BG/Q	43 (3.1x)	0.0049 (3.5x)
Aurora	600 (14x)	0.020 (4.1x)

of data streams to couple the PHASTA massively parallel computational fluid dynamics analysis package with mesh adaptation. Section 6 discusses the use of interfaces to couple the Trilinos based Albany multiphysics framework to a set of mesh services. Section 7 closes the paper.

2. COMPONENT INTERFACES

We define a software component as a set of interfaces to query and modify encapsulated state information. The components in adaptive simulations that provide geometric model, mesh, and field information [7, 18, 27] are essential to error estimation, adaptation, and load balancing [36] services. Because of this strong dependency, we provide these components and services together as the open-source (https://github.com/SCOREC/core) Parallel Unstructured Mesh Infrastructure (PUMI) [18].

The relations between the geometric model entities, mesh entities, and field tensors are required for procedures that modify the mesh topology and its distribution. For example, field tensor transfer during mesh adaptation requires the field-to-mesh relation. Likewise, the mesh-to-model relation, typically called classification, and geometric model shape information enable mesh modifications (e.g. vertex re-positioning) that are consistent with the actual geometric domain [7]. The other common interactions of the mesh with the geometric domain is in the transformation of the input field tensors onto the mesh to define the boundary conditions, material parameters and initial conditions [26].

Supporting efficient mesh modifications and field transformations requires the mesh representation used by the interfaces to provide information on any of the mesh entities and their adjacencies. Thus, PUMI implements a complete one-level mesh representation [35] to provide O(1) access.

A good example of advanced component interface usage is the PUMI Superconvergent Patch Recovery, SPR, error estimation component. The SPR routines estimate solution error by constructing an improved finite element solution using a patch-level Zienkiewicz-Zhu[41] least squares data fit. SPR provides two methods which use the patch-recovery routines. The first method recovers discontinuous solution gradients over a patch of elements and approximates an improved solution by fitting a continuous solution over the elemental patch. The second method provides an improved solution in much the same way as the first, but operates directly on integration point information obtained by the finite element analysis. The improved and primal solutions are used to create a mesh size field that drives mesh adaptation[5, 14].

3. COMPONENT INTERACTIONS

Our approaches for high performing and scalable component interactions avoid file-based I/O through in-memory data streams and component functional interfaces. The ADIOS tools provide a mechanism for the in-memory coupling of multiple executables [8, 39]. In this work, we focus on component interactions within a single executable (typically built from multiple libraries). The type of interaction, data streams or functional interfaces, chosen for a pair of components depends on their implementations.

Components that support a common file format and per-process granularity (e.g. POSIX C `stdio.h` [13] or C++ `iostream`) can use our data stream approach with minimal software changes. This approach is also a logical choice for legacy analysis codes that do not provide functional interfaces to access or create their input and output data structures. Details for implementing this approach are given in Section 4.

Components with functional interfaces that encapsulate creation, deletion, and access to underlying data structures support in-memory interactions at different levels of granularity. At a very fine level a developer may implement all mesh entity query functions such that components can share the same mesh structure; trading increased development costs for lower memory usage. An excellent example of this is the use of octree structures in the development of parallel adaptive analyses [11]. At a coarser level a developer may simply create another mesh representation through use of interfaces encapsulating mesh construction; trading higher memory usage for lower development costs. Although this method will allow for in-memory integration, it suffers from the same disadvantages as a tightly coupled approach in that a significant amount of time and effort will be required for code modification and verification. A generalization of this coarser level approach defines common sets of interfaces through which all components interact. For example, in the rotorcraft aerodynamics community the HELIOS platform provides a set of analysis, meshing, adaptation, and load balancing components via the Python-based Software Integration Framework [34].

4. DATA STREAMS

Components can pass information and avoid expensive filesystem operations through the use of buffer-based data streams. This approach is best suited for components already using POSIX C `stdio.h` [13] or C++ `iostream` String Stream APIs as the code changes required are minimal. The key changes entail passing buffer pointers during the opening and closing of the stream and adding control logic to enable stream use.

In a component using the POSIX APIs, a data stream buffer is opened with either the `stdio.h` `fmemopen` `open_memstream` or functions. `open_memstream` only supports write operations and automatically grows as needed. `fmemopen` supports reading and writing, but uses a fixed size, user specified, buffer. Once the buffer is created file read and write operations are performed through POSIX APIs accepting the `FILE` pointer returned by the buffer opening functions; i.e. `fread`, `fwrite`, `fscanf`, `fprintf`, etc After all read or write operations are complete and `fclose` is called the buffer created with `fmemopen` is automatically deallocated. A `open_memstream` created buffer requires the user to deallocate it.

The details of the C++ `iostream` APIs are left to the reader.

5. PHASTA

PHASTA (https://github.com/PHASTA/phasta) solves complex fluid flow problems [15, 32, 33, 40] on up to 768Ki cores with 3Mi MPI processes [30] using a stabilized finite element method [37] implemented with a mix of FORTRAN77 and FORTRAN90. Support for mesh adaptivity, dynamic load balancing, and reordering is provided by the C++ PUMI-based component, chef, through file I/O. This file-based coupling uses a format and procedures that were originally developed over a decade ago.

Our work adds support for PHASTA and chef in-memory adaptivity. The data stream approach for in-memory interactions was the logical choice given the existing POSIX file support, and the lack of PHASTA interfaces to modify FORTRAN data structures. The chef and PHASTA data stream implementation maintains support for POSIX file-based I/O by replacing direct calls to POSIX file open, read and write routines with function pointers.

Our work also adds a few execution control APIs to run PHASTA within an adaptive workflow. The API implementation uses the singleton design pattern and several of Miller's Smart Library techniques [25]. This approach provides backward compatibility for legacy execution modes, such as scripted file-based adaptive loops, with minimal code changes and easily accounted for the heavy reliance on global data common to legacy FORTRAN codes.

5.1 POSIX and Data Stream Performance Test

We measured the performance of PHASTA POSIX file and data stream information exchange in a adaptive analysis of incompressible flow through a rectangular channel. Tests ran on a IBM BlueGene/Q system using 16 cores per-node with a total of 256, 512, 1024, and 2048 cores. The same initial mesh with nine million elements was used for all tests. Each test executed a prescribed number of adaptive cycles. Each cycle runs the PHASTA flow solver for a fixed number of time steps, then runs size field based isotropic mesh adaptation with predictive load balancing, and ends with ParMA vertex and element balancing [36]. The adapted meshes generated by these cycles have only negligible differences across the range of core counts. As such, each core is operating on half as much data with each factor of two increase in core count.

Figure 1 depicts the average time spent per adaptive cycle reading and writing data at each core count. The time spent for the data stream approach decreases with the core count while the POSIX time increases. For data streams the time reduction is expected as the reads and writes are interacting with node-exclusive resources. POSIX time scaling has the opposite behavior. The difference in absolute time spent by the two methods is nearly proportional to the three orders of magnitude difference in the main memory and filesystem bandwidth. One non-trivial source of the POSIX slowdown is the workflows requirement to read and write the PUMI mesh in each cycle; in the data stream based workflow chef keeps the PUMI mesh in memory while the flow solver is running.

Work is underway to create a data stream based adaptive workflow for a complex flow problem with fluid-structure interactions at the boundaries of deforming solids. Figure 2 shows initial results from a multi-cycle workflow on a single solid using mesh motion and mesh adaptation.

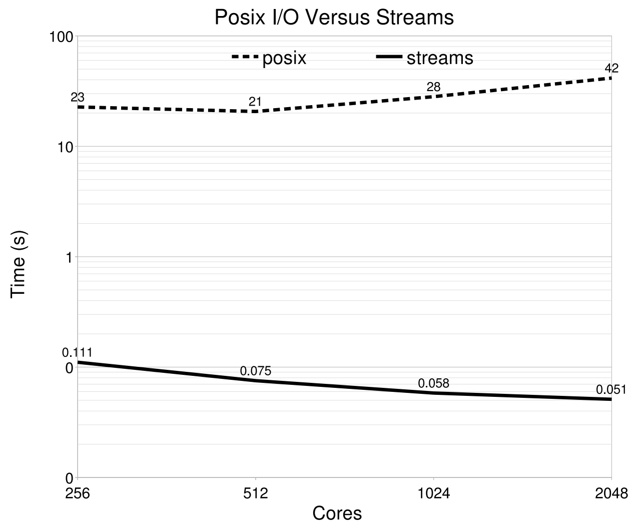

Figure 1: PHASTA POSIX I/O and data streams timings.

6. ALBANY

Albany [1] is a parallel, implicit, unstructured mesh finite element code used for the solution and analysis of partial differential equations. The code is built on over 100 software components and heavily leverages packages from the Trilinos project [2, 17]. Both Albany and Trilinos adopt an 'agile components' approach to software development that emphasizes interoperability. Albany has been used to solve a variety of multiphysics problems including ice sheet modeling and modeling the mechanical response of nuclear reactor fuel. The largest Albany runs have had over a billion degrees of freedom and used over 32Ki cores. Albany's high performance, generality, and component-based design made it an ideal candidate for the construction of an in-memory adaptive workflow.

The Albany analysis code provides an abstract base class for discretizations. Implementing the class with PUMI's full topological mesh representation simply required understanding Albany's discretization requirements. Like most standard finite element codes, Albany stores a list of mesh nodes and a node-to-element connectivity map to define mesh elements. Albany's Dirichlet and Neumann boundary conditions need additional data structures. The Dirichlet boundary condition data structure is simply an array of constrained mesh nodes. The more complex Neumann boundary condition structure requires lists of mesh elements associated with constrained mesh faces; a classification check followed by a face-to-element upward adjacency query. The PUMI implementation of Albany's discetization and boundary condition structures allows us to define and solve complex problems.

6.1 Adaptive Solderball Simulation

We ran an in-memory adaptive simulation of a solderball array subject to thermal creep. Figure 3 depicts the results of the parallel adaptive analysis using the in-memory integration of SPR and the PUMI unstructured mesh tools with

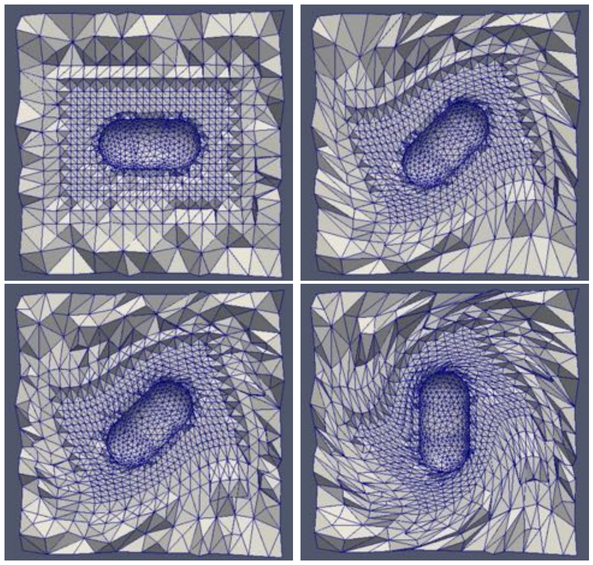

Figure 2: Multi-cycle (clockwise from top left) serial test run on rotating non-deforming capsule.

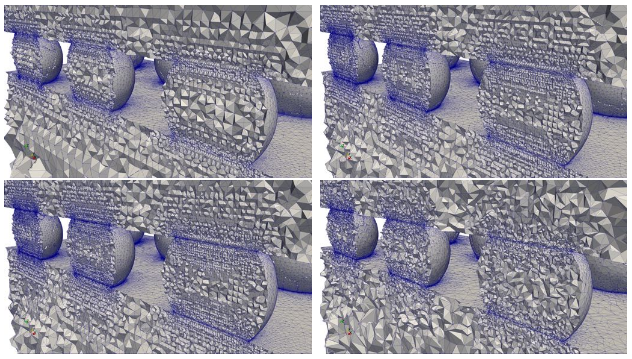

Figure 3: Four adaptation cycles (clockwise from top left) of 3x3 solderball mesh.

Albany. The adaptive workflow ran four solve-adapt cycles on 256, 512, and 1024 cores of an IBM BlueGene/Q using an initial mesh of 8M tetrahedral elements. Like the PHASTA adaptive test, the adapted meshes contain only negligible differences across the range of core counts.

The workflow begins by loading the mesh and transferring it to Albany for execution of the first analysis step. Following the analysis step, the solution information (a displacement vector at mesh nodes) and history-dependent state variables at integration points are passed in-memory to a PUMI APF FieldShape [18]. SPR then computes mesh-entity level error estimates based on an improved Cauchy stress field. The error estimate is then transformed into an isotropic mesh size filed, which the mesh adaptation software then uses to drive local mesh modification procedures. As the mesh modifications (split, collapse, etc...) are applied the FieldShape transfer operators [29, 28] are called to determine the value of state variables at repositioned or newly created integration points. Between coarsening and refinement adaptation stages, Zoltan graph methods rebalance the mesh to prevent memory exhaustion on parts where heavy refinement occurs. Once adaptation is complete ParMA rebalances the mesh to reduce element and vertex imbalances for improved linear system assembly and solve performance. The adaptive step concludes with the transformation of PUMI unstructured mesh information and APF field information into Albany analysis data structures via in-memory functional interfaces.

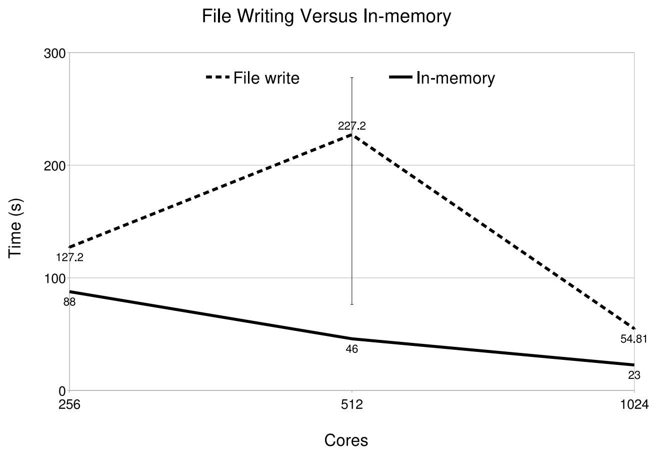

Figure 4: Average per-cycle file writing and in-memory transfer times. Minimum and maximum bars are only shown for the 512 core file writing data point where they are 6% more, or less, than the mean, respectively.

Figure 4 depicts the factor of two performance advantage of in-memory transfers of fields and mesh data between Albany, PUMI, and SPR over the writing of the mesh to POSIX files. Based on this data we estimate the performance advantage of the in-memory approach over a file-based loop that both reads and writes files to be about four times higher. Another advantage demonstrated by this data is the low in-memory transfer time imbalance; defined as maximum cycle time divided by the average cycle time. The in-memory approach has less than a 6% imbalance across all core counts while the file writing approach has a 22% imbalance at 512 cores (as shown by the large error bar in Figure 4).

7. CLOSING REMARKS

As we move towards the exascale computers being considered [12, 31], it is clear that the one of the only effective means to construct parallel adaptive simulations is by using an in-memory interfaces that avoid filesystem interactions. Of course, the cost of refactoring existing large-scale parallel partial differential equation solvers to fully interact with the type of structures and methods used by mesh adaptation components is an extremely expensive and time consuming process. To address these costs we presented approaches for in-memory integration of existing solver components with mesh adaptation components, discussed how code changes can be minimized, and demonstrated the performance advantage within adaptive simulations. In addition to efforts on developing this approach for solvers we know well (PHASTA and Albany), efforts are underway to interface other state-of-the-art solvers including NASA's FUN3D [6] and SLAC's ACE3P [22].

8. REFERENCES

[1] Albany website. http://gahansen.github.io/Albany, 2015.

[2] Trilinos website. http://trilinos.org, 2015.

[3] N. R. Adiga, G. Almasi, Y. Aridor, R. Barik, D. Beece, R. Bellofatto, G. Bhanot, R. Bickford, M. Blumrich, A. A. Bright, et al. An overview of the bluegene/l supercomputer. In *Supercomputing, ACM/IEEE 2002 Conference*, pages 60–60. IEEE, 2002.

[4] S. Alam, R. Barrett, M. Bast, M. R. Fahey, J. Kuehn, C. McCurdy, J. Rogers, P. Roth, R. Sankaran, J. S. Vetter, et al. Early evaluation of ibm bluegene/p. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, page 23. IEEE Press, 2008.

[5] F. Alauzet, X. Li, E. Seol, and M. Shephard. Parallel anisotropic 3d mesh adaptation by mesh modification. *Engineering with Computers*, 21(3):247–258, 2006.

[6] W. K. Anderson, W. D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smith. Achieving high sustained performance in an unstructured mesh cfd application. In *Proceedings of the 1999 ACM/IEEE conference on Supercomputing*, page 69. ACM, 1999.

[7] M. W. Beall, J. Walsh, and M. S. Shephard. A comparison of techniques for geometry access related to mesh generation. *Engineering with Computers*, 20:210–221, 2004.

[8] J. C. Bennett, H. Abbasi, P.-T. Bremer, R. Grout, A. Gyulassy, T. Jin, S. Klasky, H. Kolla, M. Parashar, V. Pascucci, et al. Combining in-situ and in-transit processing to enable extreme-scale scientific analysis. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–9. IEEE, 2012.

[9] M. Besta and T. Hoefler. Fault tolerance for remote memory access programming models. In *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '14, pages 37–48, New York, NY, USA, 2014. ACM.

[10] H. Bui, H. Finkel, V. Vishwanath, S. Habib, K. Heitmann, J. Leigh, M. Papka, and K. Harms. Scalable parallel i/o on a blue gene/q supercomputer using compression, topology-aware data aggregation, and subfiling. In *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, pages 107–111. IEEE, 2014.

[11] C. Burstedde, L. C. Wilcox, and O. Ghattas. `p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing*, 33(3):1103–1133, 2011.

[12] Co-Design. Crosscutting technologies for computing at the exascale, Draft Report, DOE, June, 2010. http://extremecomputing.labworks.org/crosscut/report.stm.

[13] I. C. S. P. A. S. Committee, E. Open Group (Reading, I. of Electrical, and E. Engineers. *Standard for Information Technology: Portable Operating System Interface (POSIX) : Base Specifications*. IEEE Std. 2013.

[14] P. Frey and F. Alauzet. Anisotropic mesh adaptation for cfd computations. *Computer Methods in Applied Mechanics and Engineering*, 194(48-49):5068 – 5082, 2005. Unstructured Mesh Generation.

[15] A. Y. Galimov, O. Sahni, R. T. Lahey, Jr., M. S. Shephard, D. A. Drew, and K. E. Jansen. Parallel adaptive simulation of a plunging liquid jet. *Acta Mathematica Scientia*, 30B:522–538, 2010.

[16] R. A. Haring, M. Ohmacht, T. W. Fox, M. K. Gschwind, D. L. Satterfield, K. Sugavanam, P. W. Coteus, P. Heidelberger, M. A. Blumrich, R. W. Wisniewski, et al. The ibm blue gene/q compute chip. *Micro, IEEE*, 32(2):48–60, 2012.

[17] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the trilinos project. *ACM Trans. Math. Softw.*, 31(3):397–423, 2005.

[18] D. A. Ibanez, E. S. Seol, C. W. Smith, and M. S. Shephard. Pumi: parallel unstructured mesh infrastructure. *ACM Transactions on Mathematical Software*, 2015.

[19] Intel. Aurora fact sheet. http://www.intel.com/newsroom/assets/Intel_Aurora_factsheet.pdf, 2015. Accessed: 2016-04-27.

[20] F. Isaila, J. Garcia, J. Carretero, R. Ross, and D. Kimpe. Making the case for reforming the i/o software stack of extreme-scale systems. Technical Report ANL/MCS-P5103-0314, Argonne National Laboratory, Chicago, IL, 2014.

[21] T. Z. Islam, K. Mohror, S. Bagchi, A. Moody, B. R. de Supinski, and R. Eigenmann. Mcrengine: A scalable checkpointing system using data-aware aggregation and compression. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, pages 1–11, Nov 2012.

[22] K. Ko, A. Candel, L. Ge, A. Kabel, R. Lee, Z. Li, C. Ng, V. Rawat, G. Schussman, L. Xiao, et al. Advances in parallel electromagnetic codes for accelerator science and development. *LINAC2010, Tsukuba, Japan*, pages 1028–1032, 2010.

[23] S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, and W. Allcock. I/o performance challenges at leadership scale. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, page 40. ACM, 2009.

[24] Y. J. Lo, S. Williams, B. Van Straalen, T. J. Ligocki, M. J. Cordery, N. J. Wright, M. W. Hall, and L. Oliker. Roofline model toolkit: A practical tool for architectural and program analysis. In *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, pages 129–148. Springer, 2014.

[25] M. Miller, J. Reus, R. Matzke, Q. Koziol, and A. Cheng. Smart libraries: Best sqe practices for libraries with emphasis on scientific computing. *Proceedings of the Nuclear Explosives Code Developer's Conference*, 2004.

[26] R. O'Bara, M. Beall, and M. Shephard. Attribute management system for engineering analysis. *Engineering with Computers*, 18:339–351, 2002.

[27] C. Ollivier-Gooch, L. Diachin, M. S. Shephard, T. Tautges, J. Kraftcheck, V. Leung, X.-J. Luo, and

M. Miller. An interoperable, data-structure-neutral component for mesh query and manipulation. *Transactions on Mathematical Software*, 37(3):29:1–29:28, 2010.

[28] M. Ortiz and J. Q. IV. Adaptive mesh refinement in strain localization problems. *Computer Methods in Applied Mechanics and Engineering*, 90(1-3):781 – 804, 1991.

[29] R. Radovitzky and M. Ortiz. Error estimation and adaptive meshing in strongly nonlinear dynamic problems. *Computer Methods in Applied Mechanics and Engineering*, 172(1-4):203 – 240, 1999.

[30] M. Rasquin, C. Smith, K. Chitale, E. S. Seol, B. A. Matthews, J. L. Martin, O. Sahni, R. M. Loy, M. S. Shephard, and K. E. Jansen. Scalable Implicit Flow Solver for Realistic Wing Simulations with Flow Control. *Computing in Science & Engineering*, (6):13–21, 2014.

[31] Roadmap. International exascale software project roadmap, DOE ASCR, january 2010. http://www.exascale.org/mediawiki/images/a/a1/Iesp-roadmap-draft-0.93-complete.pdf.

[32] J. M. Rodriguez, O. Sahni, K. E. Jansen, and R. T. Lahey. A parallel adaptive mesh method for the numerical simulation of multiphase flows. In *Proceedings of the 2008 Japan/US Seminar on Two-Phase Flow Dynamics*, Santa Monica, CA, Sept. 2008.

[33] O. Sahni, M. Zhou, M. S. Shephard, and K. E. Jansen. Scalable implicit finite element solver for massively parallel processing with demonstration to 160K cores. *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis - SC '09*, page 1, 2009.

[34] V. Sankaran, J. Sitaraman, A. Wissink, A. Datta, B. Jayaraman, M. Potsdam, D. Mavriplis, Z. Yang, D. O'Brien, H. Saberi, R. Cheng, N. Hariharan, and R. Strawn. Application of the helios computational platform to rotorcraft flowfields. In *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, volume 1230, pages 1–28, 2010.

[35] E. S. Seol and M. S. Shephard. Efficient distributed mesh data structure for parallel automated adaptive analysis. *Engineering with Computers*, 22(3-4):197–213, 2006.

[36] C. W. Smith, M. Rasquin, D. Ibanez, K. E. Jansen, and M. S. Shephard. Application specific partition improvement. *SIAM Journal on Scientific Computing*, submitted, 2015.

[37] C. H. Whiting and K. E. Jansen. A stabilized finite element method for the incompressible Navier-Stokes equations using a hierarchical basis. 35:93–116, 2001.

[38] H. Yu, R. K. Sahoo, C. Howson, G. Almasi, J. Castaños, M. Gupta, J. E. Moreira, J. Parker, T. Engelsiepen, R. B. Ross, et al. High performance file i/o for the blue gene/l supercomputer. In *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on*, pages 187–196. IEEE, 2006.

[39] F. Zhang, C. Docan, M. Parashar, S. Klasky, N. Podhorszki, and H. Abbasi. Enabling in-situ execution of coupled scientific workflow on multi-core platform. In *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 1352–1363. IEEE, 2012.

[40] M. Zhou, O. Sahni, H. J. Kim, C. A. Figueroa, C. A. Taylor, M. S. Shephard, and K. E. Jansen. Cardiovascular flow simulation at extreme scale. *Computational Mechanics*, 46:71–82, 2010.

[41] O. C. Zienkiewicz and J. Z. Zhu. The superconvergent patch recovery and a posteriori error estimates. part 1: The recovery technique. *International Journal for Numerical Methods in Engineering*, 33(7):1331–1364, 1992.